
Article:
Xu, Hao and Burns, Alan orcid.org/0000-0001-5621-8816 (2019) A Semi-Partitioned Model for Mixed Criticality Systems. Journal of Systems and Software. pp. 51-63. ISSN 0164-1212

https://doi.org/10.1016/j.jss.2019.01.015

A Semi-Partitioned Model for Mixed Criticality Systems

H. Xu, A. Burns

Department of Computer Science, University of York, UK

Abstract

Many Mixed Criticality algorithms have been developed with an assumption that lower criticality-level tasks may be abandoned in order to guarantee the schedulability of higher-criticality tasks when the criticality level of the system changes. But it is valuable to explore means by which all of the tasks remain schedulable through these criticality level changes. This paper introduces a semi-partitioned model for a multi-core platform that allows all of the tasks to remain schedulable if only a bounded number of cores increase their criticality level. In such a model, some lower-criticality tasks are allowed to migrate instead of being abandoned. Detailed response time analysis for this model is derived. This paper also introduces possible approaches for establishing migration routes. Together with related previous work, an appropriate semi-partitioned model for mixed criticality systems hosted on multi-core platforms is recommended.

1. Introduction and Motivation

A system containing tasks with different criticality levels is called a Mixed Criticality System (MCS). Vestal [14] was the first to introduce scheduling analysis for uni-core MCS. He introduced an algorithm that allowed all tasks, with different criticality levels, to maximise their chances of remaining schedulable regardless of changes to the system criticality mode. The system criticality mode change refers to a switch of the defined operating mode of the system, which is generally controlled by a mode change protocol [7]. In uni-core MCS, a system mode change refers to a change of the criticality level of the core; for

Email addresses: loganxuhao@126.com (H. Xu), alan.burns@york.ac.uk (A. Burns)

example, the core rises from a lower-criticality level to a higher-criticality level. Based on this notion, a variety of algorithms, such as AMC [4] and EDF-VD [3], have been developed to improve the scheduling efficiency of MCS. Most of these algorithms were defined under the assumption that there are only two criticality levels, LO-crit and HI-crit (HI-crit is of higher criticality than LO-crit). In addition, LO-crit tasks may be terminated in order to ensure the execution of HI-crit tasks when the criticality level of the system switches to HI-crit. Although these LO-crit tasks will resume execution when the system level returns to normality (i.e. LO-crit), it would be better if these tasks can remain executable throughout the system mode change. But this is not possible on single core platforms as the maximum computation capability of a core is fixed, and it is often impossible (or too expensive) to increase the performance of the core. Thus, many researchers [1] [12] [13] have started to study the problem of realising MCS on multi-core platforms; as one of the key features of a multi-core platform is that tasks may be able to migrate from one core to another, which provides more flexibility for scheduling. Moreover, in a multi-core MCS, a mode change on one core will not necessarily lead to mode changes on other cores.

Multi-core scheduling algorithms can generally be divided into three categories [9]: partitioned scheduling, global scheduling and semi-partitioned scheduling. Partitioned scheduling, where tasks are statically mapped to processors/cores, provides a stable and predictable implementation which is preferable for safety critical (i.e. HI-crit) applications. Global scheduling allows tasks to migrate from one core to another during execution, which potentially provides higher overall utilisation. Semi-partitioned scheduling is a mixture of the previous two algorithms in that hard real-time tasks may be statically mapped to cores but a small number of other tasks are able to migrate to improve schedulability. Referring to MCS, HI-crit tasks may be statically partitioned on cores in order to guarantee their execution, while some LO-crit tasks may be migrated when a system mode change is detected on their executing core.

This paper explores semi-partitioned scheduling of multi-core MCS in consideration that all tasks remain schedulable if less than a certain number of the cores enter the HI-crit mode. Following this introduction, Section 2 introduces some related work, including a piece of previous work that established the basis

of this research. Section 3 defines the essential problem of extending the semi-partitioned model to multi-core platforms and introduces the semi-partitioned model for multi-core platforms based on a probability calculation. Section 4 explores several possible task allocation models for the semi-partitioned model on a four-core platform, and recommends the most appropriate allocation model to use based on the results obtained from a comparative study. Section 5 extends the semi-partitioned model from a four-core system to a general multi-core platform. Section 6 concludes the study and discusses possible future work.

2. Related Work

A mixed-criticality system can be defined as a finite set of components. Each component has a level of criticality, L, and contains a finite set of sporadic tasks. Each task, τ_i, has a period T_i, deadline D_i, worst-case execution time C_i and criticality level L_i. There exist a number of different models to explain the relationships between the temporal attributes of the tasks and the criticality level of the tasks [8]. Vestal's model [14] assumes that the higher the degree of assurance required, the larger the task execution time needed to guarantee the completion of the task. That is, if a task τ_i has a set of assurance levels (criticality levels), $L = 1, 2, 3, 4$ with 4 being the highest, then there are four different estimations of worst-case execution time (WCET) for τ_i, with the relationships $C_i(L_1) \leq C_i(L_2) \leq C_i(L_3) \leq C_i(L_4)$.

Since tasks may have different WCET for their different criticality levels, the notation for MCS is a bit different from the standard notation for real-time systems. Table 1 shows the symbols used in this paper.

In addition, the word 'job' is used to represent one invocation/release of a 'task', and the word 'taskset' is used to represent a finite set of tasks. This paper considers the constrained-deadline sporadic task model, where the deadline of a task is smaller than or equal to its period/minimum release interval.

2.1. Response-Time Analysis

Vestal's algorithm [14] allows the priorities of high and low criticality tasks to be interleaved in order to provide flexibility in scheduling. It was later proved that Audsley's priority assignment is optimal for MCS [10].

Notation	Description
τ_i	Task i
D_i	The deadline of task τ_i
D'_i	The reduced deadline of task τ_i
T_i	The period of task τ_i
L_i	The criticality level of task τ_i
$C_i(L_i)$	The WCET of task τ_i at criticality level L_i
$U_i(L_i)$	The utilisation of task τ_i at criticality level L_i
J_i	The release jitter of task τ_i
c_j	Core j
R_i	The response time of task τ_i

Table 1: MCS Notation

Adaptive Mixed Criticality (AMC) [4] is now the standard way to analyse MCS that employ fixed priority scheduling on a uni-core, or fully partitioned multi-core, platform. With AMC a run-time monitor is used to prevent any task from executing for more than the WCET estimate it has for its own criticality level (i.e. $C_i(L_i)$), so LO-crit tasks cannot execute for more than $C(LO)$ and HI-crit tasks cannot execute for more than $C(HI)$. This monitor is also used to signal when a HI-crit task has executed for longer that its $C(LO)$ estimate.

The key notion in AMC is that lower criticality-level tasks may be suspended in order to guarantee that higher criticality-level tasks complete their executions. Assume the task set has two criticality levels: HI-crit and LO-crit. Then the schedulable test for AMC consists of three phases of analysis. The first phase is to verify the schedulability of the LO-crit mode, when all of the tasks are executing within their LO-crit budgets. The response time analysis for this phase is shown in equation (1).

$$R_i(LO) = C_i(LO) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{1}$$

where $hp(i)$ stands for the task set that contains all the tasks which have higher priority than task τ_i. This, and similar equations later in this paper, are solved using the standard techniques of forming a recurrence relation [2]. If the computed worst-case response time is no greater than the task's deadline

$(R_i(LO) \leq D_i)$ then the task is proved to be schedulable.

The second phase is to verify the schedulability of the HI-crit mode, when only HI-crit tasks are executing but they execute with their HI-crit budgets. The response time analysis for this phase is shown in equation (2) where $hpH(i)$ stands for the set of HI-crit tasks with higher priority than that of task τ_i.

$$R_i(HI) = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \tag{2}$$

Note that $R_i(HI)$ is only defined for HI-crit tasks.

The third phrase is to check the schedulability of the criticality change itself. Since exact analysis of this phase is unlikely to be tractable [4], a sufficient analysis can be done by assuming that HI-crit tasks execute with their HI-crit budget while LO-crit tasks execute with their LO-crit budget before the system changes to HI-crit mode. In this case, for a HI-crit task τ_i, interference from other HI-crit tasks will not be affected by changing the time when the system enters the HI-crit mode. But interferences from LO-crit tasks will increase if that time increases. Hence $R_i(LO)$, the time that τ_i finishes all its LO-crit budget, is the latest time the criticality change may occur. The response time for this phase $(R_i(HI)^*)$ is thus computed using equation (3) – where $hpL(i)$ stands for the set of LO-crit tasks with higher priority than that of task τ_i, and $R_i(LO)$ is a pre-computed constant (obtained from equation (1)).

$$\begin{aligned} R_i(HI)^* = & C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i(HI)^*}{T_j} \right\rceil C_j(HI) \\ & + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \end{aligned} \tag{3}$$

Note that $R_i(HI)^* \geq R_i(HI)$ so that if the task is deemed schedulable with $R_i(HI)^*$, it is deemed to be schedulable with $R_i(HI)$.

Release jitter is a key issue in modelling task execution. It occurs when a task has recurrent arrivals from a periodic source but its release event may occasionally arrive earlier than that period. Audsley et al. [2] analyse the possible interferences from tasks with the release jitter, and extend the original response time analysis equation. For example, equation (1) would become:

$$R_i(LO) = C_i(LO) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(LO) + J_j}{T_j} \right\rceil C_j(LO) \quad (4)$$

where J_j is the release jitter of task τ_j. We shall require a release jitter term in the semi-partitioned model developed below.

2.2. Semi-partitioned Model on Dual-core Platform

Xu and Burns [16] explored the semi-partitioned MCS model on a dual-core platform with two criticality levels; this study forms the basis of the work developed in this paper. They proposed a model that when a core enters the HI-crit mode, in order to keep all of the tasks schedulable, some LO-crit tasks on the core can be migrated to the other core as long as that core is still in the LO-crit mode. If both cores are in HI-crit mode, all HI-crit tasks are guaranteed to be schedulable while some LO-crit tasks are abandoned.

For the migrating tasks, it is not defined whether they have finished or partly-completed or even not yet started before the migration occurs, but all of the migrating tasks still need to execute their remaining LO-crit budget on the newly allocated core. The migrating tasks hence have reduced deadlines (D^*) after migration. To compute the exact value of such deadlines is unlikely to be tractable as all of the release patterns need to be considered; however, a sufficient analysis can be obtained by applying the shortest possible reduced deadline to each migrating task. The cited paper [16] proves that for task τ_i (which must be LO-crit), the worst-case scenario after migration is that it needs to execute all its budget in a reduced deadline of $D_i^* = D_i - (R_i(LO) - C_i(LO))$, where $R_i(LO)$ is the worst-case response time for task τ_i when the core is in LO-crit mode, and the release jitter is $J_i = R_i(LO) - C_i(LO)$. Based on this formulation, migrating tasks with relatively high priorities are likely to have smaller release jitter and smaller impact from the reduced deadline.

Xu and Burns [16] also discuss task allocation. HI-crit tasks shall be allocated first as non-migratable tasks. Considering the release jitter issue, LO-crit tasks are assigned as non-migratable tasks to the cores one by one according to a pre-sorted order (the order is discussed in the cited paper). If a task is not schedulable as a non-migratable task, two approaches are proposed: the first approach is to set this task as migratable, while the other approach is

to try to set an already allocated LO-crit higher priority task as migratable. Combined with the common bin-packing algorithms (First-Fit, Worst-Fit, and Best-Fit), there are thus six possible approaches for the semi-partitioned model on a dual-core platform (see Table 2). An experimental evaluation is reported in the paper to analyse these different approaches, as well as the base-line non-migration approach. According to the results, Semi2WF and Semi2FF have the best performance in the majority of the cases. Combined usage of Semi2WF and Semi2FF is proposed as the most appropriate method for scheduling a two criticality level MCS on a dual-core platform.

Notation	Description
Non-migration	The non-migration approach
Semi1FF	Semi-partitioned approach that migrates the fetched task and uses First Fit bin packing algorithm
Semi1BF	Semi-partitioned approach that migrates the fetched task and uses Best Fit bin packing algorithm
Semi1WF	Semi-partitioned approach that migrates the fetched task and uses Worst Fit bin packing algorithm
Semi2FF	Semi-partitioned approach that migrates the "highest" priority task and uses First Fit bin packing algorithm
Semi2BF	Semi-partitioned approach that migrates the "highest" priority tasks and uses Best Fit bin packing algorithm
Semi2WF	Semi-partitioned approach that migrates the "highest" priority tasks and uses Worst Fit bin packing algorithm

Table 2: Real-time System Notation

3. The Semi-partitioned Model for Multi-core Platform

In our previous work [16], LO-crit migratable tasks shall migrate if only one core enters the HI-crit mode while these tasks need to be abandoned if both of the cores enter HI-crit mode. Hence there exists a boundary number n_b for the semi-partitioned approach: if less than or equal to n_b cores enter the HI-crit mode, LO-crit tasks may migrate and all tasks keep executing within their corresponding criticality level budgets and meet their deadlines; otherwise, if more than n_b cores enter the HI-crit mode, only HI-crit tasks are guaranteed

their execution. At one extreme, $n-1$ can be the required boundary number for a n-core platform. However, a criticality change is expected to be a rare event and $n-1$ cores all in HI-crit mode is extremely unlikely. The determination of this boundary number n_b is an essential issue in the multi-core semi-partitioned approach. If n_b is quite small, then LO-crit tasks may need to be abandoned in many cases which is against the initial purpose of the design. If n_b is quite large, then the schedulability of the model will be quite low as the scheduling requirement becomes challenging.

We propose that the issue of setting n_b can be addressed by using a probability calculation. Assume that the probability of one core entering HI-crit mode in a sufficient long period of time is fixed (and represented by p), and the criticality mode changes on each core are independent. Based on a probabilistic argument, the probability of exactly m cores being in HI-crit mode in the same time window ($f(m,n)$) can be calculated.

$$f(m,n) = C_n^m p^m (1-p)^{n-m} = \left\{ \frac{n!}{m!(n-m)!} \right\} p^m (1-p)^{n-m} \tag{5}$$

where C_n^m represents the binomial coefficient function of choosing m out of n.

Based on this model, the probability of more than X cores entering HI-crit mode at the same time can be expressed as equation (6).

$$F(X,n) = \sum_{i=X+1}^{n} f(i,n) \tag{6}$$

According to equation (6), $F(X,n)$ will represent the probability of the case that the system needs to abandon LO-crit tasks. Hence if there exists a tolerance standard, p_{tol}, then the largest number X which meets the tolerance standard ($F(X,n) \leq p_{tol}$) can be calculated. Assuming $n_b = 2$ is a reasonable requirement in a four-core system, p_{tol} can be set as $F(2,4)$. If we further assume that $p = 10^{-4}$, an exploration is made to find out the appropriate boundary number n_b for certain sizes of platform (see Table 3).

Table 3 indicates that the boundary number n_b is much slower to rise than the number of cores. This indicates that the fault model introduced is likely to be extendable to many core platforms. However, it is also observed from the table that the exact boundary numbers n_b are irregular and hard to predict generally. We have tried to fit a curve and noted that there is a small difference

n	n_b	$F(n_b, n)$	p_{tol}
4	2	4.00E-12	4.00E-12
8	3	7.00E-15	4.00E-12
16	3	1.82E-13	4.00E-12
32	3	3.59E-12	4.00E-12
64	4	7.59E-14	4.00E-12
128	4	2.62E-13	4.00E-12
256	5	3.61E-13	4.00E-12
512	6	1.68E-13	4.00E-12
1024	7	2.67E-13	4.00E-12

Table 3: Probability Table

between $log_2 n$ and n_b (see Table 4). We therefore propose to use $log_2 n$, or more precisely $\lceil log_2(n) \rceil$, as an appropriate boundary number.

n	n_b	$log_2 n$
4	2	2
8	3	3
16	3	4
32	3	5
64	4	6
128	4	7
256	5	8
512	6	9
1024	7	10

Table 4: $log_2 n$ VS n_b

It can be proved that $\lceil log_2(n) \rceil$ is a safe boundary number (i.e. $\lceil log_2(n) \rceil \geq n_b$).

Lemma 1. *For systems with* $\{w | 2^n + 1 \leq w \leq 2^{n+1}\}$ *cores,* $n + 1$ *is a safe boundary number.*

Proof. It can be prove by induction.

- The possibility of more than K cores enter HI-crit mode in an n-core platform and an (n+1)-core platform can be represented as $F(K, n) = \sum_{i=K+1}^{n} f(i, n)$ and $F(K, n+1) = \sum_{i=K+1}^{n+1} f(i, n+1)$.
- The difference between the two functions can be viewed as: $F(K, n+1) - F(K, n) = (f(K+1, n+1) - f(K+1, n) + (f(K+2, n+1) - f(K+2), n) + ... + (f(n, n+1) - f(n, n)) + f(n+1, n+1))$.
- For each pair $f(S, n+1)$ and $f(S, n)$, they can be compared by using division. $f(S, n+1)/f(S, n) = \left\{ \frac{(n+1)!}{S!(n+1-S)!} \right\} p^S (1-p)^{n+1-S} *$

$\left\{\frac{S!(n-S)!}{n!}\right\}\frac{1}{p^S}\frac{1}{(1-p)^{n-S}} = \frac{n+1}{n+1-S} * (1-p)$.

- Since $p = 0.0001$, $(1-p) \approx 1$ and $n+1 > n+1-s$, $f(S, n+1)/f(S, n)$ shall be larger than 1.
- In that case, $f(S, n+1)$ is larger than $f(S, n)$.
- According to that, $F(K, n+1)$ is larger than $F(K, n)$, which indicates that $F(n+1, 2^{n+1})$ has the largest value in all these situations.
- Since it is shown that p_{tol} is larger than $F(n+1, 2^{n+1})$, all these situations shall fulfill the requirement.
- Thus, $n+1$ would be an appropriate boundary number for systems with $\{w|2^n+1 \leq w \leq 2^{n+1}\}$ cores.
- Since $\lceil log_2(2^n+1)\rceil = n+1$ and $\lceil log_2(2^{n+1})\rceil = n+1$, n+1 can be replaced by $\lceil log_2(NumberOfCores)\rceil$.
- In other words, $\lceil log_2(n)\rceil$ is an appropriate boundary number for systems with n cores.

□

Summing up all of the findings above, we propose a semi-partitioned model for an n-core system as following:

- If all tasks execute within their LO-crit budgets, then all deadlines are met and no task migrates.
- No LO-crit task is allowed to exceed its LO-crit budget.
- If HI-crit tasks on no more than $\lceil log_2(n)\rceil$ cores exceed their LO-crit budgets, then some LO-crit tasks will migrate, but *all* LO-crit tasks and HI-crit tasks remain schedulable. These migrating tasks will be divided and migrate to paired cores that are currently in LO-crit mode.
- If HI-crit tasks on more than $\lceil log_2(n)\rceil$ cores exceed their LO-crit budgets (but are within their HI-crit bugdet ($C(HI)$), then some LO-crit tasks will be abandoned, but all HI-crit tasks remain schedulable (without migration).

Tasks that have migrated return to their host core for new job releases if this core has returned to the LO-crit mode. Such a mode change will occur if there is an idle tick (or indeed instance) on that core.

4. Semi-partitioned model on Four-core Platform

As the boundary number issue has been addressed and the model has been redefined for the multi-core platform, the next step is to solve the migration destination problem. In a dual-core platform [16], migratable tasks have only one core to migrate to. But in a multi-core platform, tasks may literally migrate to any possible other core, which may cause the whole system to become unpredictable and hard to analyse. This section will explore this migration destination issue on a four-core platform. It will first introduce three allocation models and provide a brief exploration upon the working mechanisms of these models, as well as a comparison of the models based on response-time analysis. An evaluation of the models will be given at the end of this section.

4.1. Migration Models

For a four-core platform, if only one core enters HI-crit mode, the migrating tasks have three possible cores to migrate towards. We propose three models based on the distribution of these migrating tasks.

- Model 1 represents the model in which all migrating tasks migrate to one core.
- Model 2 represents the model in which all migrating tasks migrate to two cores.
- Model 3 represents the model in which all migrating tasks migrate to three cores.

The following subsections will introduce the details of these models, including the relationship among cores and how the migratable tasks are divided to migrate to different cores. Note that for this 4-core system, mode changes on less than or equal to two cores must be tolerated without loss of schedulability.

4.1.1. Model 1

In Model 1, migratable tasks on one core may only migrate to one available core through a fixed route. The model can be viewed in Figure 1, where the rectangles stand for cores and arrows stand for migration routes.

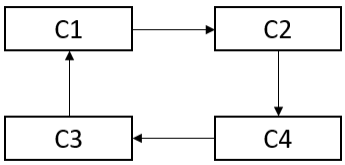

Figure 1: Model 1

According to the figure, the migration routes form a circle which indicates that it is always possible to find an available core (a core still in LO-crit mode) following the routes. Figures 2 and Figure 3 indicate two example scenarios for Model 1. A core in grey indicates that this core is currently in HI-crit mode; the thinner arrows indicate a set of tasks migrating from one core to another; the thick arrow indicates different steps of the scenarios (the left hand side of the arrow is step 1 while the right hand side is step 2).

Based on these scenarios, the migrating load seems to be the main issue of Model 1. In Step 2 of Scenario 1 and Scenario 2, an extremely heavy task load is migrated to c_4 while no task migrates to c_3 which will undoubtedly affect the schedulability of the model.

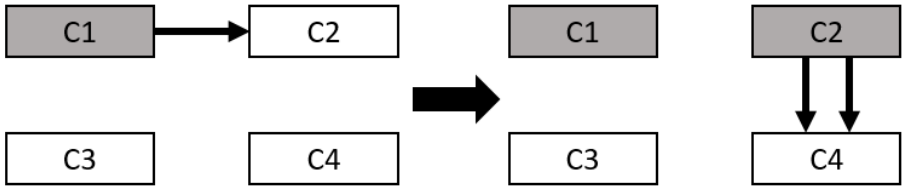

Figure 2: Model 1 Scenario 1

Model 1: Scenario 1 (Figure 2)

1. Core c_1 enters HI-crit mode, all of the migratable tasks on c_1 will migrate to c_2.
2. Core c_2 enters HI-crit mode, all of the migratable tasks on c_2, including tasks migrated from c_1, will migrate to c_4.

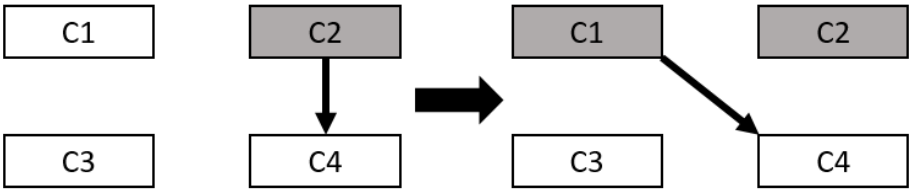

Figure 3: Model 1 Scenario 2

Model 1: Scenario 2 (Figure 3)

1. Core c_2 enters HI-crit mode, all of the migratable tasks on c_2 will migrate to c_4.
2. Core c_1 enters HI-crit mode, all of the migratable tasks on c_1 will migrate to c_2. But since c_2 is already in HI-crit mode, these migrating tasks will migrate to c_4 directly.

4.1.2. Model 2

This model allows migratable tasks to migrate to two cores rather than one. This leads to two issues: how to decide which two cores to migrate to and how the migratable tasks shall be divided. With regard to the first issue, Model 2 pairs the cores into four groups: (c_1, c_2), (c_1, c_3), (c_2, c_4) and (c_3, c_4). Each core has two partner cores and migrating tasks originally on the core will only migrate to the partner cores if they are available. For example, c_1 is paired with c_2 and c_3. If c_1 enters HI-crit mode, all of the migrating tasks on c_1 will migrate to c_2 and c_3. The model can be viewed in Figure 4.

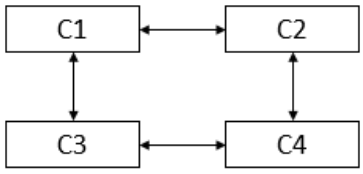

Figure 4: Model 2

Model 2: Scenario 1 (Figure 5)

1. Core c_1 enters HI-crit mode, migratable tasks on c_1 will split into two groups and migrate to c_2 due to the pairing relationship (c_1, c_2) and c_3 due to the pairing relationship (c_1, c_3).

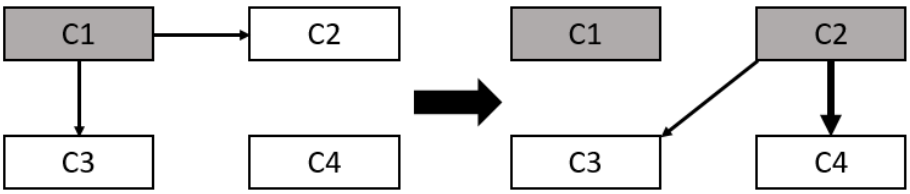

Figure 5: Model 2 Scenario 1

2. Core c_2 now enters HI-crit mode, all of the migratable tasks originally on c_2 will migrate to c_4 due to the pairing relationship (c_2, c_4), and all of the tasks that migrate from c_1 will attempt to migrate back to c_1. But since c_1 is already in HI-crit mode, these tasks will migrate to c_3. (In practice, these tasks will directly migrate to c_3.)

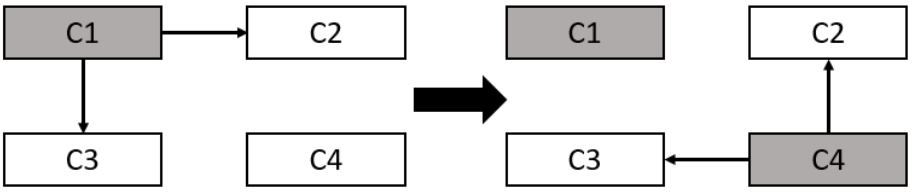

Figure 6: Model 2 Scenario 2

Model 2: Scenario 2 (Figure 6)

1. Core c_1 enters HI-crit mode, all of the migratable tasks on c_1 will split into two groups and migrate to c_2 and c_3.
2. Core c_4 now enters HI-crit mode, all of the migratable tasks on c_4 will split into two groups and migrate to c_2 and c_3.

The splitting of tasks into the two groups is undertaken using the WF bin-packing algorithm as this provides more balanced task distribution than FF or BF. The following model also uses WF.

4.1.3. Model 3

Model 3 is a quite different model from the previous ones. In this model, tasks are allowed to migrate to all of the cores currently in the LO-crit mode. Figure 7 depicts the migration paths that Model 3 allows for.

As Figure 7 shows, migratable tasks can migrate to all of the cores which makes for maximum usage of the computational capacity of the system. Here is a possible scenario for Model 3:

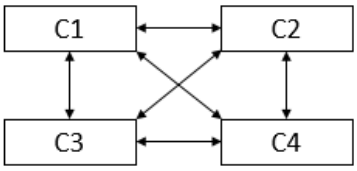

Figure 7: Model 3

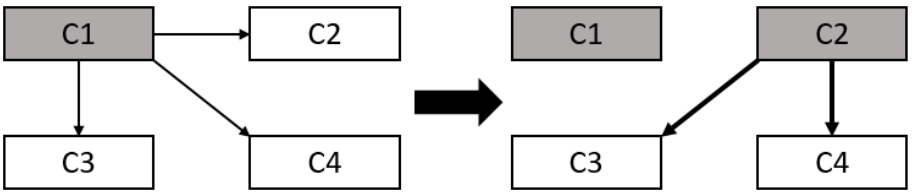

Figure 8: Model 3 Scenario 1

Model 3: Scenario 1 (Figure 8)

1. Core c_1 enters HI-crit mode, all of the migratable tasks on c_1 will be split into three groups and migrate to all other cores in LO-crit mode (c_2, c_3 and c_4).
2. Core c_2 now enters HI-crit mode, all of the migratable tasks on c_2 will be split into two groups and migrate to the other cores in LO-crit mode (c_3 and c_4). The tasks that migrated from c_1 to c_2 are also migrated to c_3 and c_4.

4.2. Model Analysis

The previous section has introduced three possible allocation models. This section will give a detailed exploration of the models proposed with their corresponding response-time schedulability analysis.

4.2.1. Model 1

In this model, cores are chained in a circle. Assume the platform contains four cores (c_1, c_2, c_3 and c_4), then cores shall be chained as $c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4 \rightarrow c_1$. If c_1 enters its HI-crit mode, then some LO-crit tasks on c_1 migrate to core c_2. If core c_2 also enters HI-crit mode, then these migrated tasks from c_1 shall also migrate to c_3.

To define this model, assume that a taskset S contains tasks with two criticality levels (HI-crit and LO-crit). If this taskset is to be scheduled on a four-core

platform by Model 1, then on each core there shall exist three types of tasks: HI-crit tasks, statically allocated LO-crit tasks and migratable LO-crit tasks. Let HI_i represent the set of HI-crit tasks on c_i, LO_i represent the set of statically allocated LO-crit tasks and $MIG_{i,j,k}$ represent the LO-crit tasks that can migrate from c_i to c_j, and then possibly to c_k: $i \rightarrow j \rightarrow k$. The following relationship can be obtained:

- $S = (LO_1 \cup LO_2 \cup LO_3 \cup LO_4) \cup (HI_1 \cup HI_2 \cup HI_3 \cup HI_4)$
  $\cup (MIG_{1,2,3} \cup MIG_{2,3,4} \cup MIG_{3,4,1} \cup MIG_{4,1,2})$

In the steady state mode, all these tasks are statically partitioned on each core and executing with their LO-crit budgets. Define state X_m to represent this phase, then the relationships between tasks and cores can be viewed as:

- $X_1 = LO_1 \cup HI_1 \cup MIG_{1,2,3}$
- $X_2 = LO_2 \cup HI_2 \cup MIG_{2,1,4}$
- $X_3 = LO_3 \cup HI_3 \cup MIG_{3,1,4}$
- $X_4 = LO_4 \cup HI_4 \cup MIG_{4,2,3}$
- $S = X_1 \cup X_2 \cup X_3 \cup X_4$

In each state X_m, all tasks are executing within their LO-crit budgets. In this case, the response time analysis of all tasks is given by equation (7):

$$\forall \tau_i \in X_m : R_i(LO) = C_i(LO) + \sum_{\tau_j \in chp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{7}$$

where $chp(i)$ is the set of tasks with priority greater than τ_i that execute on the same core as τ_i (i.e. c_m).

If a criticality change occurs on one core (c_i), then HI-crit tasks (HI_i) will in the worst-case execute with their HI-crit budgets. For LO-crit tasks, some of them (LO_i) still execute on the core with their LO-crit budgets while the others ($MIG_{i,j,k}$) need to migrate to another core as there is not enough scheduling space for them on the core. Define state $Y(i)_m$ to represent the state of core c_m when core c_i enters its HI-crit mode – here tasks in $MIG_{i,j,k}$ will be migrated from c_i to c_j and the relationship between tasks and cores is given by:

- $Y(1)_1 = LO_1 \cup HI_1$
- $Y(1)_2 = X_2 \cup MIG_{1,2,3}$
- $Y(1)_3 = X_3$
- $Y(1)_4 = X_4$
- $S = Y(1)_1 \cup Y(1)_2 \cup Y(1)_3 \cup Y(1)_4$

With this state, the behaviour of the cores is quite similar to the semi-partitioned model analysed for the dual-core platform. According to that, the reduced deadlines and release jitter need to be applied to the migrating tasks, and the worst case is given, for example, by equation (8):

$$\begin{aligned} &\forall \tau_i \in MIG_{1,2,3} : \\ &\qquad D'_i = D_i - (R_i(LO) - C_i(LO)) \\ &\qquad J_i = R_i(LO) - C_i(LO) \end{aligned} \tag{8}$$

Thus, the response time of tasks on core c_1 and core c_2 ($R_i(MIX)$) in this state is given by equation (9):

$$\begin{aligned} &\forall \tau_i \in Y(1)_1 : \\ &R_i(MIX) = C_i(L_i) + \sum_{\tau_j \in chp(i)} \left\lceil \frac{R_i(MIX)}{T_j} \right\rceil C_j(L_j) \\ &\qquad + \sum_{\tau_k \in chpMIG(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \\ &\forall \tau_i \in Y(1)_2 : \\ &R_i(LO)' = C_i(LO) + \sum_{\tau_j \in chp(i)} \left\lceil \frac{R_i(LO)' + J_j}{T_j} \right\rceil C_j(LO) \end{aligned} \tag{9}$$

where L_i is the criticality of the task under consideration (τ_i), and $chpMIG(i)$ is the set of LO-crit tasks on the same core as τ_i which migrate away from that core if there is a criticality mode change.

This equation will be used in subsequent analysis for different sets of tasks.

If a second core enters HI-crit mode, there exists two different scenarios. The first scenario is that the core, which has not accepted any migrated tasks,

enters HI-crit mode. For example, if core c_3 enters HI-crit mode, then all of the migratable tasks on core c_3 shall migrate to core c_4 due to the chained relationship. The relationship between tasks and cores can be viewed as below, where $Y(a,b)_c$ represents the state of c_c when c_a enters HI-crit mode first and c_b enters its HI-crit mode later:

- $Y(1,3)_1 = Y(1)_1$
- $Y(1,3)_2 = Y(1)_2$
- $Y(1,3)_3 = LO_3 \cup HI_3$
- $Y(1,3)_4 = X_4 \cup MIG_{3,4,1}$
- $S = Y(1,3)_1 \cup Y(1,3)_2 \cup Y(1,3)_3 \cup Y(1,3)_4$

The response time analysis equation for c_3 and c_4 in this state is the same as that of c_1 and c_2 in the previous state Y_1.

The second scenario is that the core, which accepts migrated tasks, then enters HI-crit mode. In this scenario, not only the migratable tasks on the core, but also the accepted migrated tasks need to migrate to another core. If c_2 enters HI-crit mode, then all of the migratable tasks on core c_2 shall migrate to core c_3, and the relationship between tasks and cores can be viewed as:

- $Y(1,2)_1 = Y(1)_1$
- $Y(1,2)_2 = LO_2 \cup HI_2$
- $Y(1,2)_3 = X_3 \cup MIG_{1,2,3} \cup MIG_{2,3,4}$
- $Y(1,2)_4 = X_4$
- $S = Y(1,2)_1 \cup Y(1,2)_2 \cup Y(1,2)_3 \cup Y(1,2)_4$

With this scenario, taskset $MIG_{1,2,3}$ migrates a second time. Since the migration progress will cause the reduced deadline and the release jitter effects, a task migrating a second time may suffer a further reduction in deadline and increase in release jitter. In other words, the release jitter and the reduced deadline effects are compounded. It is observed that the worst case happens when a task migrates to one core and further migrates to another core in one

release. For instance, τ_i waits a maximum time $(R_{i,m} - C_i)$[1] before it starts to execute on c_m and then migrates to c_n. Then it waits another maximum time $(R_{i,n} - C_i)$ before it starts to execute on c_n and migrates to core c_o. Thus, for these tasks, the worst case of release jitters and reduced deadlines is given by equation (10):

$$\begin{aligned} &\forall \tau_i \in MIG_{1,2,3} : \\ &\qquad D_i'' = D_i - (R_{i,m} - C_i) - (R_{i,n} - C_i) \\ &\qquad J_i' = (R_{i,m} - C_i) + (R_{i,n} - C_i) \end{aligned} \tag{10}$$

However, despite the changes to deadlines and release jitter of the tasks migrating a second time, the response time analysis for the other tasks remains the same as that in the other scenario.

If further cores enter HI-crit mode, then all of LO-crit tasks on that core need to be abandoned as the number of cores in HI-crit mode exceeds the boundary number. For example, if core c_3 enters HI-crit mode, then the relationship between tasks and cores can be described as:

- $Y(1,2,3)_1 = Y(1)_1$
- $Y(1,2,3)_2 = Y(1,2)_2$
- $Y(1,2,3)_3 = HI_3$
- $Y(1,2,3)_4 = X_4$

Based on this state view, only HI-crit tasks are executing on c_3 while all of the LO-crit tasks are abandoned. The response time analysis of c_3 in this state is given by equation (11):

$$\begin{aligned} &\forall \tau_i \in Y(1,2,3)_3 : \\ &\qquad R_i(HI)' = C_i(HI) + \sum_{\tau_j \in chph(i)} \left\lceil \frac{R_i(HI)'}{T_j} \right\rceil C_j(HI) \\ &\qquad\qquad + \sum_{\tau_k \in chpL(i)} \left\lceil \frac{R_i(LO)' + J_k'}{T_k} \right\rceil C_k(LO) \end{aligned} \tag{11}$$

[1] As τ_i is of low criticality it only has one worst-case execution time $R_i(LO)$ and one computation time $C_i(LO)$, the designation '(LO)' is therefore omitted from equation (10).

4.2.2. Model 2

This model pairs the cores so that tasks may only migrate between paired cores. The four pairs are: (c_1, c_2), (c_3, c_4), (c_1, c_3) and (c_2, c_4). If c_1 enters its HI-crit mode, then LO-crit tasks on core c_1 may migrate to either c_2 or c_3 but not c_4. Assume that all of the migratable tasks on c_1 have migrated to c_2, if c_2 also enters HI-crit mode then tasks, which previously migrated to c_2, will have to migrate to c_3, while all of the migrating tasks originally on c_2 will migrate to c_4. Based on this rule, the schedulability test of this model can be simplified into several dual-core semi-partitioned models, which is simpler than the previous model.

We use the same notation as Model 1 to give an initial partitioning of the taskset:

- $S = (LO_1 \cup LO_2 \cup LO_3 \cup LO_4) \cup (HI_1 \cup HI_2 \cup HI_3 \cup HI_4)$
  $\cup (MIG_{1,2,3} \cup MIG_{1,3,2}) \cup (MIG_{2,1,4} \cup MIG_{2,4,1})$
  $\cup (MIG_{3,1,4} \cup MIG_{3,4,1}) \cup (MIG_{4,2,3} \cup MIG_{4,3,2})$

In the steady state mode, all these tasks are statically partitioned on each core and executing within their LO-crit budgets. Define state X_m to represent this phase, then the relationship between tasks and cores can be viewed as:

- $X_1 = LO_1 \cup HI_1 \cup MIG_{1,2,3} \cup MIG_{1,3,2}$
- $X_2 = LO_2 \cup HI_2 \cup MIG_{2,1,4} \cup MIG_{2,4,1}$
- $X_3 = LO_3 \cup HI_3 \cup MIG_{3,1,4} \cup MIG_{3,4,1}$
- $X_4 = LO_4 \cup HI_4 \cup MIG_{4,2,3} \cup MIG_{4,3,2}$
- $S = X_1 \cup X_2 \cup X_3 \cup X_4$

In state X_m, all tasks are executing with their LO-crit budgets. In this case, the response time analysis of all tasks is given by the earlier equation (7).

If a criticality change occurs on one core (say c_1), then HI-crit tasks (HI_1) will execute with their HI-crit budgets. For LO-crit tasks, some of them (LO_1) still execute on the core with their LO-crit budgets, while the others need to migrate to other cores. Following the construction given earlier:

- $Y(1)_1 = LO_1 \cup HI_1$

- $Y(1)_2 = X_2 \cup MIG_{1,2,3}$
- $Y(1)_3 = X_3 \cup MIG_{1,3,2}$
- $Y(1)_4 = X_4$
- $S = Y(1)_1 \cup Y(1)_2 \cup Y(1)_3 \cup Y(1)_4$

In state $Y(1)$ where only core c_1 enters HI-crit mode, HI-crit tasks on this core will execute with their HI-crit budgets while LO-crit staying tasks will execute with their LO-crit budgets. All of the tasks on other cores will still execute with their LO-crit budgets. Reduced deadlines and release jitter will be applied to migrating tasks, and the worst case is again given by equation (8) for tasksets $MIG_{1,2,3}$ and $MIG_{1,3,2}$. The response time analysis of this state is given by equation (9) for tasks in $Y(1)_1$, $Y(1)_2$ and $Y(1)_3$.

If another core enters HI-crit mode, there are two possible scenarios: the core which receives migrated tasks enters its HI-crit mode, and the core which does not have any migrated tasks enters its HI-crit mode. In the first case, assume core c_4 enters HI-crit mode in state $Y(1)$, then HI-crit tasks on c_4 will execute with their HI-crit budgets and migratable LO-crit tasks will migrate to c_2 and core c_3. The relationship between tasks and cores can be viewed as:

- $Y(1,4)_1 = Y(1)_1$
- $Y(1,4)_2 = Y(1)_2 \cup MIG_{4,2,3}$
- $Y(1,4)_3 = Y(1)_3 \cup MIG_{4,3,2}$
- $Y(1,4)_4 = LO_4 \cup HI_4$
- $S = Y(1,4)_1 \cup Y(1,4)_2 \cup Y(1,4)_3 \cup Y(1,4)_4$

In this scenario, the response time analysis is similar to the previous state $Y(1)$, which will not be repeated.

In the second scenario, assume c_2 enters HI-crit mode in state $Y(1)$, then HI-crit tasks on c_2 will execute with their HI-crit budgets, migratable LO-crit tasks that originally were allocated on c_2 will all migrate to c_4 as c_1 is already in HI-crit mode, and the migratable LO-crit tasks previously migrated from c_1 will migrate to c_3. The relationship between tasks and cores is as follows:

- $Y(1,2)_1 = Y(1)_1$
- $Y(1,2)_2 = LO_2 \cup HI_2$
- $Y(1,2)_3 = X_3 \cup MIG_{1,3,2} \cup MIG_{1,2,3}$
- $Y(1,2)_4 = X_4 \cup MIG_(2,1,4) \cup MIG_{2,4,1}$
- $S = Y(1,2)_1 \cup Y(1,2)_2 \cup Y(1,2)_3 \cup Y(1,2)_4$

In this situation all task subsets $Y(i,j)$: c_j, which has not accepted any migrated tasks, enters HI-crit mode after c_i has entered HI-crit mode. After that, HI-crit tasks on this core will execute with their HI-crit budgets while LO-crit staying tasks will execute with their LO-crit budgets. All of the tasks on other cores will still be executing with their LO-crit budgets. Reduced deadlines and release jitter will be applied to the migrating tasks, and in this case only migrating tasks originally from core c_i will suffer from further reduced deadlines and release jitter. Equation (8) again shows the worst case for taskset $MIG_{1,2,3}$.

The response time analysis of core c_2, core c_3 and core c_4 in state $Y(i,j)$ is given by the equivalent of equation (9).

If further cores enter HI-crit mode, then both migratable and non-migratable LO-crit tasks on the mode changing core need to be abandoned to guarantee the execution of HI-crit tasks as the number of cores in HI-crit mode exceeds the boundary number. Assume core c_3 enters HI-crit mode in state $Y(1,2)$, then the relationship between tasks and cores can be viewed as:

- $Y(1,2,3)_1 = Y(1)_1$
- $Y(1,2,3)_2 = LO_2 \cup HI_2$
- $Y(1,2,3)_3 = HI_3$
- $Y(1,2,3)_4 = X_4 \cup MIG_(2,1,4) \cup MIG_{2,4,1}$

In this state, only HI-crit tasks on core c_3 are executing with their HI-crit budgets while all migratable and migrated LO-crit tasks on the core are abandoned. The response time analysis of core c_3 in this state is again given by equation (11).

If all of the response times for all possible states are no greater that the corresponding deadlines then the taskset is deemed to be schedulable.

4.2.3. Model 3

In this model, tasks are allowed to migrate to any possible core for maximum flexibility. When only one core enters HI-crit mode, then some LO-crit tasks may stay on the core executing with their LO-crit executing budgets while some other LO-crit tasks will migrate to other cores. In this model, these migrating LO-crit tasks will be allocated "equally" to all cores. This "equally" here not only represents the number of tasks but also needs to consider the sum of the utilization of the migratable tasks on each core. If another core also enters HI-crit mode, then some LO-crit tasks, which are originally executing on the core, may stay on the core executing with their LO-crit executing budgets, while some other LO-crit tasks and the LO-crit tasks migrated to the core will migrate to other cores which are in LO-crit mode. These migrating tasks will also migrate "equally". If a further core enters HI-crit mode, then all of the LO-crit tasks on the core will be abandoned in order to guarantee the execution of the HI-crit tasks.

Again assume that a taskset S contains several tasks in two criticality levels (HI-crit and LO-crit) and that this taskset is split into the three sets on each of the four cores: HI_i, LO_i and $MIG_{i,j,k}$. Then the following relationship can be obtained:

- $S = (LO_1 \cup LO_2 \cup LO_3 \cup LO_4) \cup (HI_1 \cup HI_2 \cup H_3 \cup H4)$
  $\cup((MIG_{1,2,3} \cup MIG_{1,2,4}) \cup (MIG_{1,3,2} \cup MIG_{1,3,4}) \cup (MIG_{1,4,2} \cup MIG_{1,4,3}))$
  $\cup((MIG_{2,1,3} \cup MIG_{2,1,4}) \cup (MIG_{2,3,1} \cup MIG_{2,3,4}) \cup (MIG_{2,4,1} \cup MIG_{2,4,3}))$
  $\cup((MIG_{3,1,2} \cup MIG_{3,1,4}) \cup (MIG_{3,2,1} \cup MIG_{3,2,4}) \cup (MIG_{3,4,1} \cup MIG_{3,4,2}))$
  $\cup((MIG_{4,1,2} \cup MIG_{4,1,3}) \cup (MIG_{4,2,1} \cup MIG_{4,2,3}) \cup (MIG_{4,3,1} \cup MIG_{4,3,2}))$

In the steady state mode, all these tasks are statically partitioned on each core and executing with their LO-crit budgets. Using X and Y is the same way in the others two models we obtain:

- $X_1 = LO_1 \cup HI_1 \cup (MIG_{1,2,3} \cup MIG_{1,2,4}) \cup (MIG_{1,3,2} \cup MIG_{1,3,4}) \cup (MIG_{1,4,2} \cup MIG_{1,4,3})$
- $X_2 = LO_2 \cup HI_2 \cup (MIG_{2,1,3} \cup MIG_{2,1,4}) \cup (MIG_{2,3,1} \cup MIG_{2,3,4}) \cup (MIG_{2,4,1} \cup MIG_{2,4,3})$

- $X_3 = LO_3 \cup HI_3 \cup (MIG_{3,1,2} \cup MIG_{3,1,4}) \cup (MIG_{3,2,1} \cup MIG_{3,2,4}) \cup (MIG_{3,4,1} \cup MIG_{3,4,2})$
- $X_4 = LO_4 \cup HI_4 \cup (MIG_{4,1,2} \cup MIG_{4,1,3}) \cup (MIG_{4,2,1} \cup MIG_{4,2,3}) \cup (MIG_{4,3,1} \cup MIG_{4,3,2})$
- $S = X_1 \cup X_2 \cup X_3 \cup X_4$

Again response-time equation (7) can be used for this steady state mode.

- $Y(1)_1 = LO_1 \cup HI_1$
- $Y(1)_2 = X_2 \cup (MIG_{1,2,3} \cup MIG_{1,2,4})$
- $Y(1)_3 = X_3 \cup (MIG_{1,3,2} \cup MIG_{1,3,4})$
- $Y(1)_4 = X_4 \cup (MIG_{1,4,2} \cup MIG_{1,4,3})$
- $S = Y(1)_1 \cup Y(1)_2 \cup Y(1)_3 \cup Y(1)_4$

Reduced deadlines and release jitter are applied to migrating tasks as in the earlier models. Thus, the response time analysis for state $Y(i)$ is given by the equivalent of equation (9).

If a further criticality change occurs on core c_j then all of the migratable LO-crit tasks on this core need to migrate to other cores while all of the HI-crit tasks execute with their HI-crit budgets. For example:

- $Y(1,2)_1 = LO_1 \cup HI_1$
- $Y(1,2)_2 = LO_2 \cup HI_2$
- $Y(1,2)_3 = Y(1)_3 \cup (MIG_{2,3,1} \cup MIG_{2,3,4}) \cup MIG_{2,1,3} \cup MIG_{1,2,3}$
- $Y(1,2)_4 = Y(1)_4 \cup (MIG_{2,4,1} \cup MIG_{2,4,3}) \cup MIG_{2,1,4} \cup MIG_{1,2,4}$
- $S = Y(1,2)_1 \cup Y(1,2)_2 \cup Y(1,2)_3 \cup Y(1,2)_4$

Within this state tasksets $MIG_{1,2,3}$ and $MIG_{1,2,4}$ will migrate a second time. As discussed in Model 1, further reduced deadlines and release jitter will be applied to these migrating tasks. The response time analysis for cores c_2, c_3 and c_4 in this state is again given by equation (9).

If more cores enter HI-crit mode, then only HI-crit tasks on these cores will remain executing while all migratable and migrated LO-crit tasks on these cores need to be abandoned. Assume core c_3 enters HI-crit mode, then the relationship between tasks and cores can be viewed as:

- $Y(1,2,3)_1 = LO_1 \cup HI_1$
- $Y(1,2,3)_2 = LO_2 \cup HI_2$
- $Y(1,2,3)_3 = HI_3$
- $Y(1,2,3)_4 = Y(1)_4 \cup (MIG_{2,4,1} \cup MIG_{2,4,3}) \cup MIG_{2,1,4} \cup MIG_{1,2,4}$

Based on the state, all migrating LO-crit tasks on core c_3 are abandoned while HI-crit tasks are executing within their HI-crit budgets. The response time analysis for the core in this state is again given by equation (11).

If all of the response times for all possible states are no greater that the corresponding deadlines then the taskset is deemed to be schedulable.

4.3. Evaluation of the Models

The previous section has derived sufficient response-time analysis for all of the allocation models introduced. In this section, we will undertake an evaluation to compare the scheduling efficiency of the allocation models. At the end of this section a recommended approach is proposed.

4.3.1. Experiment Configuration

In order to explore the effectiveness of the three models, a comparative evaluation was undertaken. Software was produced to compare the performances of Model 1, Model 2 and Model 3. The software consists of three parts. The first part of the software generates tasksets. Tasks are randomly set to be HI-crit tasks or LO-crit tasks but the percentage of HI-crit tasks is controlled to be a fixed number (named "Percent" and symbolised as "P" in this section of the paper). In addition, for all HI-crit tasks, their HI-crit WCETs are calculated by using a parameter (named "Factor" and symbolised as "f"); this is used to describe the multiple rate between HI-crit WCET and LO-crit WCET. The values of "Percent", "Factor" and the number of tasks in each taskset will be changed in the experiment to explore the performance of the three models over a wide range of task parameters.

In order to gain uniform distributed parameters, UUnifast-discard algorithm [6] is used to generate 'nominal' utilization (a 'nominal' utilization represents the LO-crit utilization for a LO-crit task or the HI-crit utilization for a HI-crit task), and Log-uniform algorithm [11] is used to generate periods. Other parameters of each task can be calculated based on these two values ($D = T, C(L_i) = U_i(L_i) * T$ – for $C(L_i)$ equal to $C(LO)$ and $C(HI)$.

The second part of the software is to pre-sort each taskset before allocation and scheduling. As stated in the task allocation section, all tasks will be sorted in descending criticality-aware utilization order. In such an order, all HI-crit tasks will be placed in front of all LO-crit tasks.

The last part of the software contains the response time analysis introduced in Section 4.2 and explores the scheduling success rate of the three models.

4.3.2. Results and Comparison

We investigate the performance of Model 1, Model 2 and Model 3 on a four core platform and compare them with the non-migration algorithm. The non-migration algorithm is chosen as a lower bound on performance. Figure 9 shows the percentage of the tasksets that are schedulable for a system of 24 tasks (where half of the tasks are HI-crit, the criticality factor is 2, $P = 0.5$ and $f = 2$). The Y-axis shows the percentage of the successfully scheduled tasksets while the X-axis shows the sum of nominal utilizations of the tested taskset. The sum of utilizations ranges from 3.2 to 4.6 in steps of 0.028 – this range covers the significant behaviours of the models.

From Figure 9, it can be observed that all of the models outperform the non-migration one by a considerable margin. For example, as shown by the black lines, Model 3 can schedule around 75% of the tasksets when the taskset utilization is around 3.7, while the non-migration model can only schedule around 57% of the tasksets. The improvement in schedulability of Model 3 over non-migration is about $\frac{75-57}{57} * 100\% = 31.58\%$, which is significant. Comparing all of the semi-partitioned methods, Model 3 has the best performance, but the difference between Model 3 and Model 2 is not large.

In order to explore the performance of the algorithms relating to criticality factor ($C(HI)/C(LO)$) and the percentage of HI-crit tasks, weighted schedulability measurement is used [5]. Weighted schedulability measure $W_y(p)$ is used

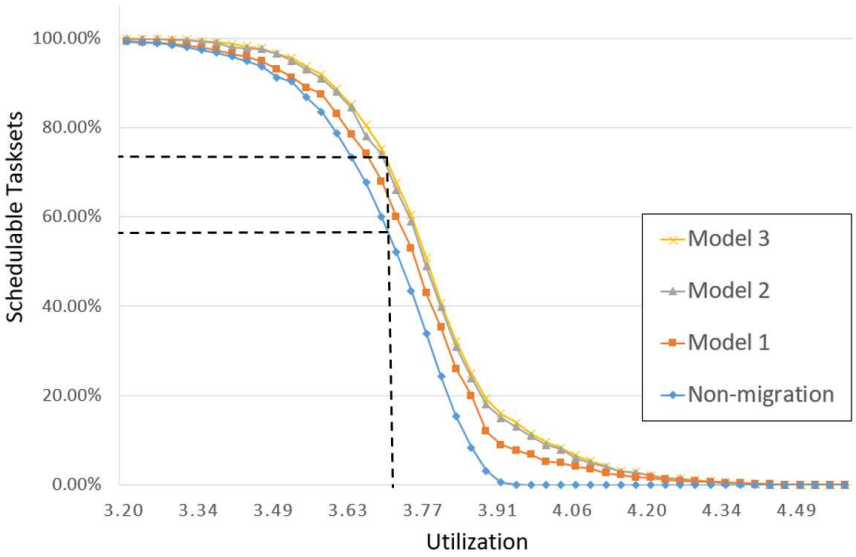

Figure 9: Percentage of Schedulable Tasksets on a 4-core Platform

for schedulability test y as a function of parameter p to reduce a 3-dimensional plot to 2 dimensions:

$$W_y(p) = (\sum_{\forall \Gamma} u(\Gamma) * S_y(\Gamma, p))/\sum_{\forall \Gamma} u(\Gamma) \tag{12}$$

In this situation equation (12), for each value of p, it combines results for all of the tasksets Γ generated for all of a set of equally spaced utilization levels (same as that in previous figure, 3.2 to 4.6 in steps of 0.028). $S_y(\Gamma, p)$ is the binary result (1 or 0) of schedulability test y for a taskset Γ with parameter value p while $u(\Gamma)$ represents the utilization of taskset Γ.

We show how the results are changed by varying one key parameter at a time. Figure 10 varies the criticality factor, Figure 11 varies the percentage of HI-crit tasks and Figure 12 varies the size of the taskset. The X-axis stands for the parameter examined and the Y-axis represents the weighted value. According to Figure 10, Model 3 has the best performance, while Model 2 provides slightly less schedulability. In addition, both models have increased performance as the criticality factor increases. This is to be expected as the increase of WCET difference between different criticality levels allows more scheduling potential for the migrating tasks.

With regard to Figure 11, the performance of the semi-partitioned algorithms

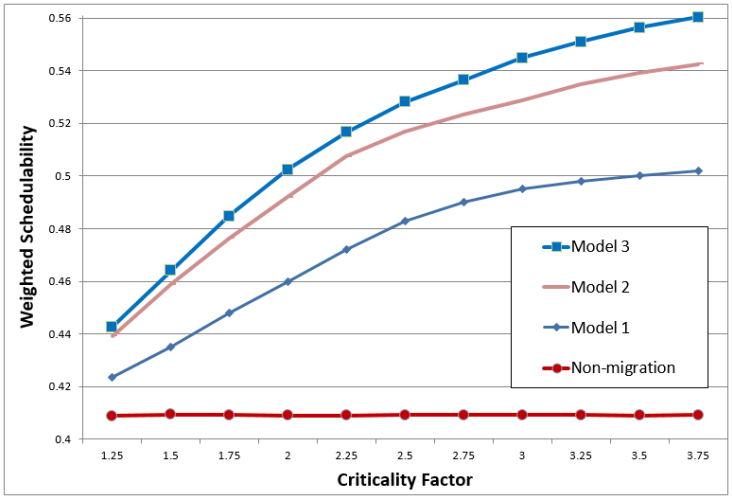

Figure 10: Varying the Criticality Factor

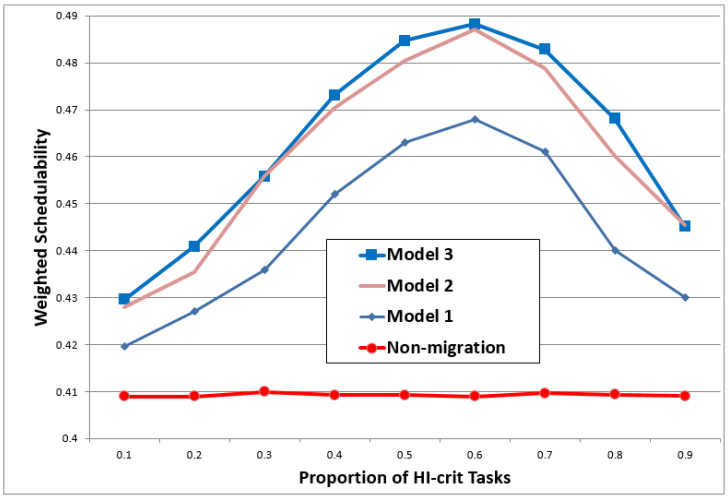

Figure 11: Varying the Criticality Percent

has formed an inverted U-shape curve since each end of the interval represents a one-criticality taskset. The individual performance of Model 3 has the best performance while Model 2 provides slightly less schedulability. In addition, it is observed that the difference between Model 3 and Model 2 decreases when the percentage of the criticality tasks is approaching 0.6.

Figure 12, also contains an inverted U-shape curve. This is expected as tasks are relatively large in smaller sized tasksets which adds difficulty in finding migratable tasks, while in larger sized tasksets, the interference from high priority

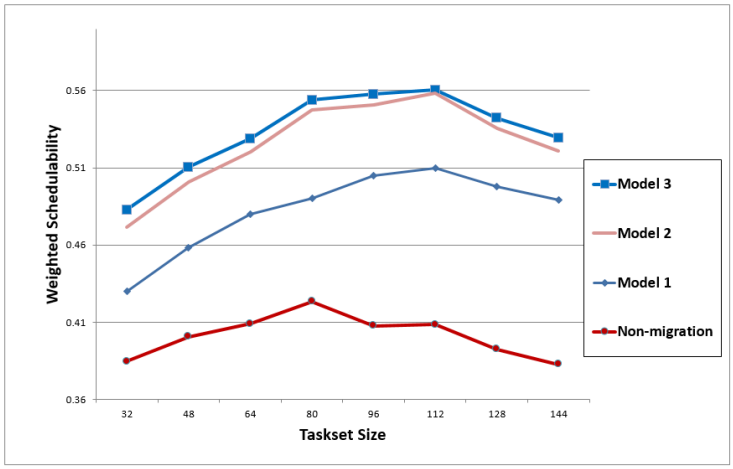

Figure 12: Varying the Taskset Size

tasks increases, as well as the effects from release jitter, which adds difficulty to the schedulability of migrated tasks with reduced deadlines. In terms of the individual performance, Model 3 still has the best performance while the performance of Model 2 is slightly poorer.

4.4. Recommend Approach

Overall, it is observed that Model 3 provides the best schedulability in all cases. However, the schedulability difference between Model 3 and Model 2 is not significant. As it has been argued that Model 3 has much more complex scheduling analysis, Model 2 is suggested to be the most appropriate model for a 4-core MCS with two criticality levels. It is also the one recommended for general n-core platforms.

5. Extending to n-core Platforms

The previous section has indicated that Model 2, which pairs the cores into groups, is the most suitable and scalable migration model for multi-core platforms. This section will discuss how to extend Model 2 to an n-core platform. It will first show two detailed examples of extending the migration model to an 8-core platform (which represents the approach when n is even) and a 7-core platform (which represents the case when n is odd). Finally, a general extendable scheme is defined.

5.1. An 8-core Platform Example

In a 4-core platform, the boundary number is 2 and each core has two paired cores. In addition, migratable tasks are split into two sets when migrating. Based on this information, it is reasonable to assume that for a 8-core platform, since the boundary number is 3, each core shall have three paired cores and migratable tasks shall be split into three sets when migrating[2]. Thus, the problem becomes: how are the cores to be paired to fulfil the above requirements. We propose a clone algorithm to solve the problem.

- Assume that there exists a 4-core platform (c_1, c_2, c_3, c_4) and cores are paired into four pairs $(c_1, c_2), (c_1, c_3), (c_2, c_4), (c_3, c_4)$ as previously in Model 2.
- Create a second clone 4-core platform (c'_1, c'_2, c'_3, c'_4) and pair the cores in the same way $(c'_1, c'_2), (c'_1, c'_3), (c'_2, c'_4), (c'_3, c'_4)$.
- Pair the original cores with the clone cores: $(c_1, c'_1), (c_2, c'_2), (c_3, c'_3), (c_4, c'_4)$.
- Replace c'_1 by c_5, c'_2 by c_6, c'_3 by c_7 and c'_4 by c_8 in all of the pairs generated.

According to the algorithm, we can obtain the pairing relationship for a 8-core platform as following: (c_1, c_2), (c_1, c_3), (c_2, c_4), (c_3, c_4), (c_5, c_6), (c_5, c_7), (c_6, c_8), (c_7, c_8), (c_1, c_5), (c_2, c_6), (c_3, c_7), (c_4, c_8), which fulfills the requirement that each core is paired with three different cores. In addition, by using the same algorithm iteratively, we can extend the pairing relationship to 16-core platforms, 32-core platforms, ..., and thus any 2^n-core platform.

5.2. A 7-core Platform Example

A 7-core platform is a special case. According to the semi-partitioned model definition, the boundary number for this platform is still 3 but it is not possible to pair the cores so that each core has three different paired partners. This can be proved by contradiction as follows:

1. Assume there exists a pairing method to pair 7 cores so that each core is paired to three different cores.

[2]Three pairs ensures that there is at least one partner that is not in the HI-crit mode – as the core in question is moving to HI-crit mode, and the boundary number is three, there can be at most two others cores already in that mode.

2. There exist $3 \times 7 = 21$ relationships between the cores.
3. Pair one core to another always results in 2 relationships.
4. There does not exist a possibility to create an odd number of pairing relationships. Contradiction found.
5. Therefore, 7 cores cannot be paired into groups so that each core has three different paired cores.

As the boundary number calculated by $\lceil log_2(n) \rceil$ is, in general, slightly larger than the exact boundary number, it is acceptable to make one core have a smaller boundary number (2 in this scenario) and only pair it with two cores. Thus, a modified clone algorithm can be used to solve the pairing problem for a 7-core platform.

- Assume that there exists a 4-core platform (c_1, c_2, c_3, c_4) and cores are paired into four pairs $(c_1, c_2), (c_1, c_3), (c_2, c_4), (c_3, c_4)$ as previously in Model 2.
- Create a cloned 4-core platform (c'_1, c'_2, c'_3, c'_4) and pair the cores in the same way $(c'_1, c'_2), (c'_1, c'_3), (c'_2, c'_4), (c'_3, c'_4)$.
- Pair the original cores with the clone cores: $(c_1, c'_1), (c_2, c'_2), (c_3, c'_3), (c_4, c'_4)$.
- Replace c'_1 by c_5, c'_2 by c_6, c'_3 by c_7 and c'_4 by c_8 in all of the pairs generated.
- Delete all of the pairing relationships with core c_8: $(c_6, c_8), (c_7, c_8), (c_4, c_8)$.
- For each two deleted pairing relationship, create a new pairing relationship between two different cores excluding core c_8: (c_6, c_7).

According to this algorithm, we can get the pairing relationship for a 7-core platform as following: (c_1, c_2), (c_1, c_3), (c_2, c_4), (c_3, c_4),(c_5, c_6), (c_5, c_7), (c_1, c_5), (c_2, c_6), (c_3, c_7), (c_6, c_7). With this pairing relationship, for core c_1, c_2, c_3, c_5, c_6, c_7, each has three paired partners, while core c_4 has only two.

5.3. General Algorithm

Assuming an n-core platform, with the boundary number of $\lceil log_2(n) \rceil$, there exist two different situations: n is an even number or n is an odd number. If n is an even number, then there exists an integer k such that $n = 2 \times k$.

In order to apply Model 2, the cores in the system require to be paired so that each core has $\lceil log_2(n) \rceil$ paired partners. By applying the clone algorithm, the pairing relationship of the n-core platform can be generated by finding the pairing relationship of a k-core platform where the boundary number is $\lceil log_2(n) \rceil - 1 = \lceil log_2(2 \times k) \rceil - 1 = log_2(2) + \lceil log_2(k) \rceil - 1 = \lceil log_2(k) \rceil$. Thus, finding the pairing relationship of a k-core platform will solve the pairing problem for this n-core platform.

If n is an odd number, then there exists an integer k' such that $n = 2 \times k' - 1$. Similar to the previous scenario, in order to apply Model 2, the cores in the system require to be paired so that most cores have $\lceil log_2(n) \rceil$ paired partners. By applying the modified clone algorithm, the pairing relationship of the n-core platform can be generated by finding the pairing relationship of a k'-core platform where the boundary number is $\lceil log_2(n) \rceil - 1 = \lceil log_2(2 \times k' - 1) \rceil - 1 = log_2(2) + \lceil log_2(k') \rceil - 1 = \lceil log_2(k') \rceil$. Thus, finding the pairing relationship of a k'-core platform will solve the pairing problem for this n-core platform. In all, the pairing relationship of an n-core platform can be generated by the recursive usage of the clone algorithm and the modified clone algorithm. When the pairing relationship between cores is derived, migratable tasks may be split into the boundary number of groups and migrate to the paired cores when required by the system's run-time behaviour.

6. Conclusion

This paper has explored the semi-partitioned MCS model on a multi-core platform. It first addresses the boundary number determination problem by the use of a probability calculation. In consideration of easier usage, it is proposes that $\lceil log_2(n) \rceil$ shall be used as the boundary number for an n-core system. That is, for an n-core system, all tasks shall remain schedulable as long as no more than $\lceil log_2(n) \rceil$ cores enter the HI-crit mode. This paper then explores the task allocation problem on a four-core platform. Three task allocation models are proposed and analysed by response time analysis and evaluated via a set of experiments.

According to the results observed and the consideration of calculation complexity, it is suggested that Model 2, which splits the migratable task load within

paired cores, will be the most appropriate task allocation model for a four-core system. In addition, this paper has provided an iterative algorithm to obtain a possible pairing relationship for an n-core platform following Model 2.

In our previous work [16], we illustrate that the combined usage of Semi2WF and Semi2FF (which we termed just Semi2) provides the best scheduling performance for a dual-core platform. In other words, when the migration source core and the migration destination core are fixed, the Semi2 algorithm is an appropriate approach to determine which tasks are candidates for migration. Based on that, we propose an appropriate semi-partitioned model for an n-core system as:

- Each core is paired with $\lceil log_2(n) \rceil$ other cores.
- The Semi2 approach is used to determine which LO-crit tasks shall be designated migratable.
- If c_i enters HI-crit mode and the total number of the cores in HI-crit mode is no more than $\lceil log_2(n) \rceil$, migratable tasks on c_i migrate 'equally' (by the offline use of the WF bin-packing algorithm) to the paired cores which are still in LO-crit mode. All tasks are guarantee to meet their deadlines.
- If c_i enters HI-crit mode and the total number of the cores in HI-crit mode would be more than $\lceil log_2(n) \rceil$, all LO-crit tasks on c_i will be abandoned/suspended. Only HI-crit tasks are guaranteed to meet their deadlines.

Future work on this approach will incorporate the overheads of migration into the analysis. An initial study on this topic demonstrates that the semi-partitioned MCS model remains viable even when realistic overheads are included [15].

Acknowledgements

The research that went into writing this paper is funded in part by the ESPRC grant MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.

7. References

[1] J. H. Anderson, S. K. Baruah, and B. Brandenburg. Multicore operating-system support for mixed criticality. In *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*. Citeseer, 2009.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–22. IEEE, 2010.

[4] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[5] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. *Proceedings of OSPERT*, pages 33–44, 2010.

[6] E. Bini and G.C. Buttazzo. Measuring the performance of schedulability tests. *Journal of Real-Time Systems*, 30(1-2):129–154, 2005.

[7] A. Burns. System mode changes-general and criticality-based. In *Workshop on Mixed Criticlity Systems*, pages 3–8. RTSS, 2014.

[8] A. Burns and R.I. Davis. A survey of research into mixed criticality systems. *ACM Computer Surveys*, 50(6):1–37, 2017.

[9] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4):35, 2011.

[10] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-time systems*, 46(3):305–331, 2010.

[11] P. Emberson, R. Staggord, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools*

and Methodologies for Embedded and Real-time System (WATERS), pages 6–11, 2010.

[12] O. R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *Trust, Security and Privacy in Computing and Communications*, pages 1051–1059. IEEE, 2011.

[13] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens. Multi-criteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors. In *Workshop on Mixed Criticality Systems*, 2013.

[14] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

[15] H. Xu. *A Semi-Partitioned Model for Scheduling Mixed-Criticality Multi-Core Systems*. Ph.D. thesis, Department of Computer Science, University of York, UK, 2017.

[16] H. Xu and A. Burns. Semi-partitioned model for dual-core mixed criticality system. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pages 257–266. ACM, 2015.